Advanced Mockito Hands-On Exercises

Exercise 1: Mocking Databases and Repositories

You need to test a service that interacts with a database repository.

Steps:

1. Create a mock repository using Mockito.
2. Stub the repository methods to return predefined data.
3. Write a test to verify the service logic using the mocked repository.

```
4 usages
public interface Repository {
    2 usages
    String getData();
}
```

```
2 usages
public class Service {
    2 usages
    private final Repository repository;

    1 usage
    public Service(Repository repository){
        this.repository = repository;
    }

    1 usage
    public String processData(){
        return "processed "+repository.getData();
    }

}
```

```
✓ ServiceTest (com.examp 1 sec 606 ms
    ✓ testServiceWithMock 1 sec 606 ms
✓ Tests passed: 1 of 1 test – 1 sec 606 ms
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
```

Exercise 2: Mocking External Services (RESTful APIs)

You need to test a service that calls an external RESTful API.

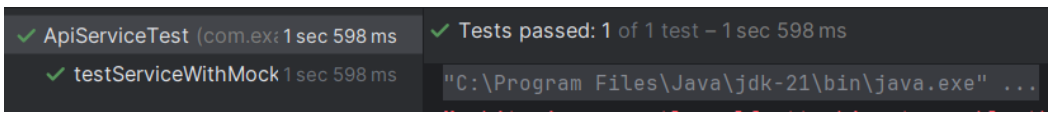
Steps:

1. Create a mock REST client using Mockito.
2. Stub the REST client methods to return predefined responses.
3. Write a test to verify the service logic using the mocked REST client.

```
4 usages
public class RestClient {
    2 usages
    public String getResponse() {
        return "Real API";
    }
}
```

```
public class ApiService {
    2 usages
    private final RestClient restClient;

    1 usage
    public ApiService(RestClient restClient) {
        this.restClient = restClient;
    }

    1 usage
    public String fetchData() {
        return "Fetched " + restClient.getResponse();
    }
}
```

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

public class ApiServiceTest {
    @Test
    public void testServiceWithMockRestClient() {
        RestClient mockClient = mock(RestClient.class);
        when(mockClient.getResponse()).thenReturn( t: "Mock Response");

        ApiService service = new ApiService(mockClient);
        String result = service.fetchData();

        assertEquals( expected: "Fetched Mock Response", result);
    }
}
```



Exercise 3: Mocking File I/O

You need to test a service that reads from and writes to files.

Steps:

1. Create a mock file reader and writer using Mockito.
2. Stub the file reader and writer methods to simulate file operations.
3. Write a test to verify the service logic using the mocked file reader and writer.

```java
public interface FileReader {
    2 usages
    String read();
}
```

```java
public interface FileWriter {
    2 usages
    void write(String content);
}
```

```java
    1 usage
    public FileService(FileReader reader, FileWriter writer) {
        this.reader = reader;
        this.writer = writer;
    }

    1 usage
    public String processFile() {
        String content = reader.read();
        writer.write( content: "Processed " + content);
        return "Processed " + content;
    }
}
```

```java
public class FileServiceTest {
    @Test
    public void testServiceWithMockFileIO() {
        FileReader mockReader = mock(FileReader.class);
        FileWriter mockWriter = mock(FileWriter.class);

        when(mockReader.read()).thenReturn( t: "Mock File Content");

        FileService fileService = new FileService(mockReader, mockWriter);
        String result = fileService.processFile();

        assertEquals( expected: "Processed Mock File Content", result);
        verify(mockWriter).write( content: "Processed Mock File Content");
    }
}
```

```
✓ FileServiceTest (com.ex 1 sec 544 ms
  ✓ testServiceWithMock 1 sec 544 ms
✓ Tests passed: 1 of 1 test – 1 sec 544 ms
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
```

Exercise 4: Mocking Network Interactions

You need to test a service that interacts with network resources.

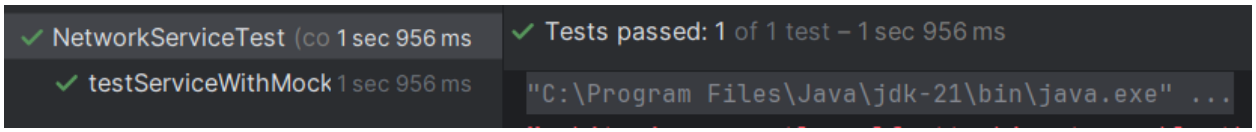
Steps:

4. 1. Create a mock network client using Mockito.
5. 2. Stub the network client methods to simulate network interactions.
6. 3. Write a test to verify the service logic using the mocked network client.

```
public interface NetworkClient {
    2 usages
    String connect();
}
```

```
3   public class NetworkService {
        2 usages
4       private final NetworkClient client;
5
        1 usage
6       public NetworkService(NetworkClient client) {
7           this.client = client;
8       }
9
        1 usage
10      public String connectToServer() {
11          return "Connected to " + client.connect();
12      }
13  }
```

```
5   import static org.junit.jupiter.api.Assertions.*;
6   import static org.mockito.Mockito.mock;
7   import static org.mockito.Mockito.when;
8
9   public class NetworkServiceTest {
10      @Test
11      public void testServiceWithMockNetworkClient() {
12          NetworkClient mockClient = mock(NetworkClient.class);
13          when(mockClient.connect()).thenReturn( t: "Mock Connection");
14
15          NetworkService service = new NetworkService(mockClient);
16          String result = service.connectToServer();
17
18          assertEquals( expected: "Connected to Mock Connection", result);
19      }
20  }
```



Exercise 5: Mocking Multiple Return Values

You need to test a service that calls a method multiple times with different return values.

Steps:

1. Create a mock object using Mockito.
2. Stub the method to return different values on consecutive calls.
3. Write a test to verify the service logic using the mocked object.

```
public interface Repository {
    3 usages
    String getData();
}
```

```
public class Service {
    2 usages
    private final Repository repository;

    2 usages
    public Service(Repository repository) { this.repository = repository; }

    3 usages
    public String processData() { return "Processed " + repository.getData(); }
}
```

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

public class MultiReturnServiceTest {
    @Test
    public void testServiceWithMultipleReturnValues() {
        Repository mockRepo = mock(Repository.class);
        when(mockRepo.getData())
                .thenReturn( t: "First Mock Data")
                .thenReturn( t: "Second Mock Data");
        Service service = new Service(mockRepo);
        String first = service.processData();
        String second = service.processData();
        assertEquals( expected: "Processed First Mock Data", first);
        assertEquals( expected: "Processed Second Mock Data", second);
    }
}
```

```
✓ MultiReturnServiceTest ( 1 sec 591 ms
  ✓ testServiceWithMultip 1 sec 591 ms
✓ Tests passed: 1 of 1 test – 1 sec 591 ms
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
```